



Java* Performance Profiling using the VTune™ Performance Analyzer

Intel® VTune™ Performance Analyzer
Intel VTune Performance Analyzer for Linux*

Introduction

The Intel® VTune™ Performance Analyzer has been used to tune C/C++ applications for years and the support the analyzer provides is fairly well known, but many developers don't realize that most of that support also applies to Java* applications. The VTune analyzer has the ability to identify performance bottlenecks in source code, locate performance-sensitive CPU events, and display the flow of control in Java applications.

The two primary technologies used in the analysis are Call Graph and Time- and Event-Based Sampling. Call Graph enables the developer to profile the Java application, identify performance bottlenecks, and possibly locate parts of the application that use the Java API inefficiently. Time- and Event-Based Sampling technology helps locate application hotspots — areas that consume significant amounts of time, based upon operating system timer events or specific processor events. This information allows the developer to characterize the performance of various Java Virtual Machines (JVMs) on a particular platform, as well as to tune any platform-dependent code not related to the JVM. The VTune analyzer's unique mixed-mode profiling feature enhances a developer's ability to tune applications that mix Java and C/C++ code using the Java Native Interface (JNI). This paper provides an overview and a usage example of these technologies.

Behind the Scenes

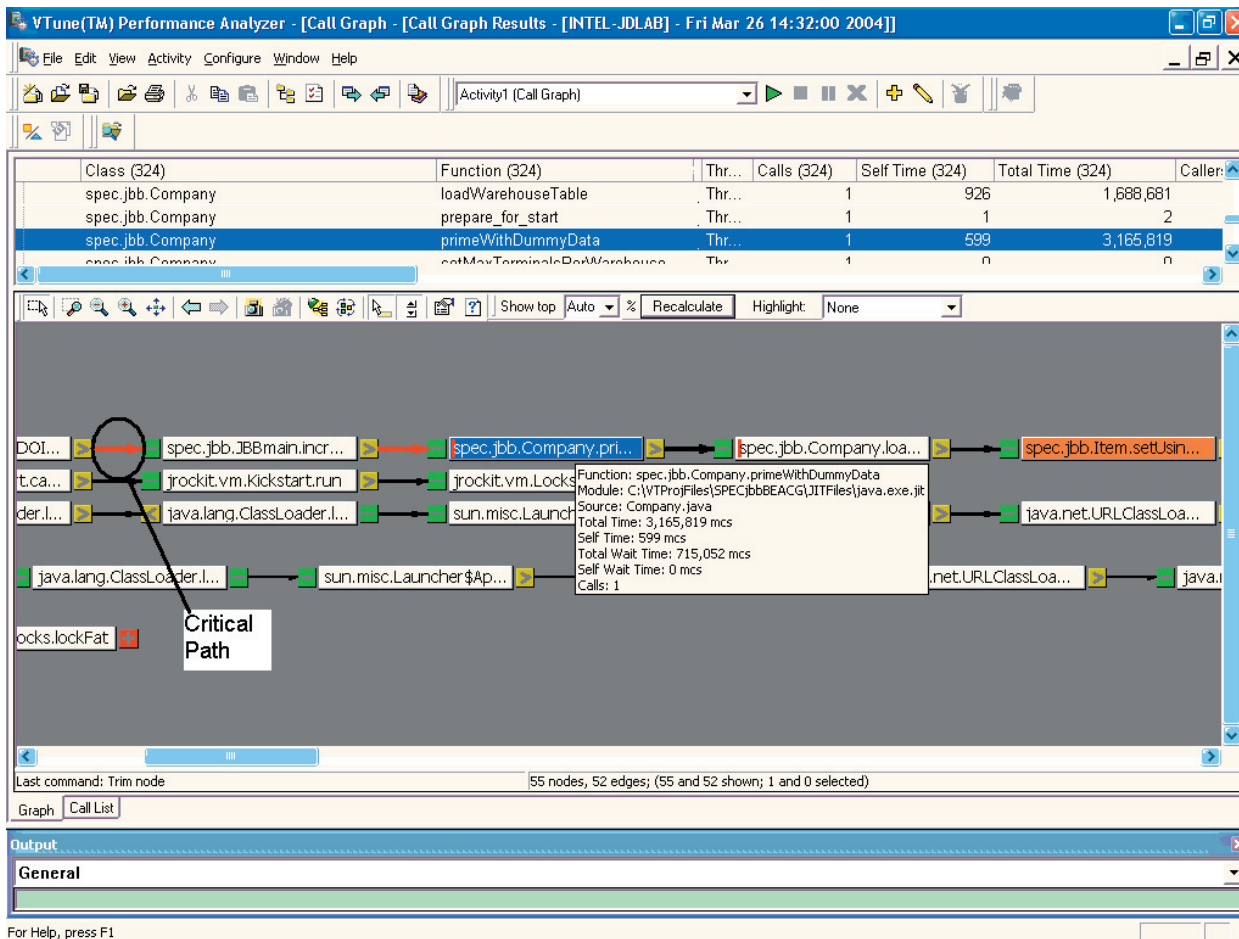
The VTune analyzer uses an industry standard interface into the JVM, the JVMPI (JVM Profiling Interface), for gathering Java-specific information. This interface communicates data to the VTune analyzer: memory locations and method names of JIT (just-in-time) emitted code, calls between Java methods, symbol information, etc. Virtually any current JVM will work with the VTune analyzer if it implements the standard JVMPI interface.

Getting the Big Picture — Program Flow of Control with Call Graph

The Call Graph technology of the VTune analyzer gathers and displays information related to the method-by-method flow of control through the Java program. This technology not only helps identify the methods that consume the most execution time, it also shows the calling sequences that invoked those methods, and identifies the critical path through the program. It provides an algorithmic view, locating the areas in the application that consume a significant amount of time, with an indication of how often particular methods are called and the amount of time spent waiting for called methods. This data enables a developer to identify the specific methods and algorithms that are candidates for performance tuning.

For Java applications, Call Graph is enabled by APIs that exist between the VTune analyzer and the JVM. The JVM sends a message to the VTune analyzer with each method call. As a result, there may be some performance impact when gathering Call Graph data. As with Time- and Event-Based Sampling, the VTune analyzer's Call Graph feature can show the complete flow graph of mixed applications, where Java functions call native functions via the JNI, and vice versa. This mixed mode Call Graph feature is currently available only for the Microsoft Windows* operating systems. Figure 1 is an example of the information the VTune analyzer Call Graph technology provides. The RED arrow at the top left corner of the graph identifies the critical path, or the path that consumes the most time in the application. This is the path where the Java developer may find good tuning candidates. Call Graph uses color coding to indicate relative time each method takes to execute in relation to all other methods within the application. The dark orange items in the Call Graph are items that have a significant amount of execution time. Hovering the mouse over a particular method shows detailed information about the method. In this example, information is shown for `spec.jbb.Company.PrimeWithDummyData` in the critical path.

Figure 1 - Example of Information Provided by Call Graph Technology



Although the “Total Time” the method consumes is significant, the method is only called once (as indicated by the “Calls” value), and most of the time is spent waiting for methods that spec.jbb.Company.PrimeWithDummyData calls (as the “Total Wait Time” value indicates). The developer should move farther down the critical path to identify other methods in the call tree, looking for those that consume a significant amount of time (“Total Time”), where most of the time is spent in the method’s code (“Self Time”), and it is called many times (“Calls”). Once a method is identified that meets this criteria, it is a candidate for further analysis. For parts of the code that developers have access to and that are platform specific, they can use Time-Based Sampling to “drill down” to the specific parts of the code that are causing the performance bottlenecks. Developers can use Event-Based Sampling to sample on specific processor events; however, this feature is more useful in mixed-mode applications, and less useful in pure Java applications.

Thus, Call Graph provides an application profile. It provides the “big picture” and identifies the hotspots. Once hotspots and the critical path are identified, the developer is able to analyze the algorithms and the Java APIs used, and determine whether or not further analysis is warranted. If the hotspot is in an area where the developer has access to the source code, it can be investigated using another algorithm, or investigated using a different Java API. If the bottleneck is in the JVM, the developer can investigate using a different JVM, optimized for the specific platform, or can work with the JVM vendor to optimize the JVM for the platform.

Identifying Performance Bottlenecks – Time- and Event-Based Sampling

We have seen how the Call Graph technology profiles an application and gives the developer clues to locate performance bottlenecks at the application level. The Time- and Event-Based Sampling technology provides information about system-wide performance, based upon user-selected events. A monitored event could be as general as the “Clockticks” event or the operating system timer, or it could be specific to a platform architecture, such as the “1st level cache load misses retired” event for the Intel® Pentium® 4 processor. If the source code is available, the VTune analyzer will annotate each executable statement with performance data related to these events, allowing the developer to easily identify those parts of the program with performance issues. This technology also relies on JVMPI that communicates runtime JIT compiler operations.

One of the biggest advantages to the Time- and Event-Based Sampling technology is that it is non-intrusive and generates very little overhead. When the Java application is running, the VTune analyzer generates occasional hardware interrupts after a user-specified number of monitored events occurs. At the time of the interrupt, the VTune analyzer saves the current CPU execution context and uses this data later during the post analysis phase to generate annotated Java source code and other performance graphs. The VTune analyzer is essentially “sampling” the CPU execution context using CPU or operating system timer events to trigger the samples. Since the VTune analyzer is both a C/C++ and Java profiling tool, it provides this type of performance information for mixed Java and native code applications as well as for pure Java applications. Once the samples have been gathered, if source code is available, the VTune analyzer allows the developer to drill down to the specific lines of source that generated the largest number of event samples. The developer

Figure 2 - Example of the “Modules” View

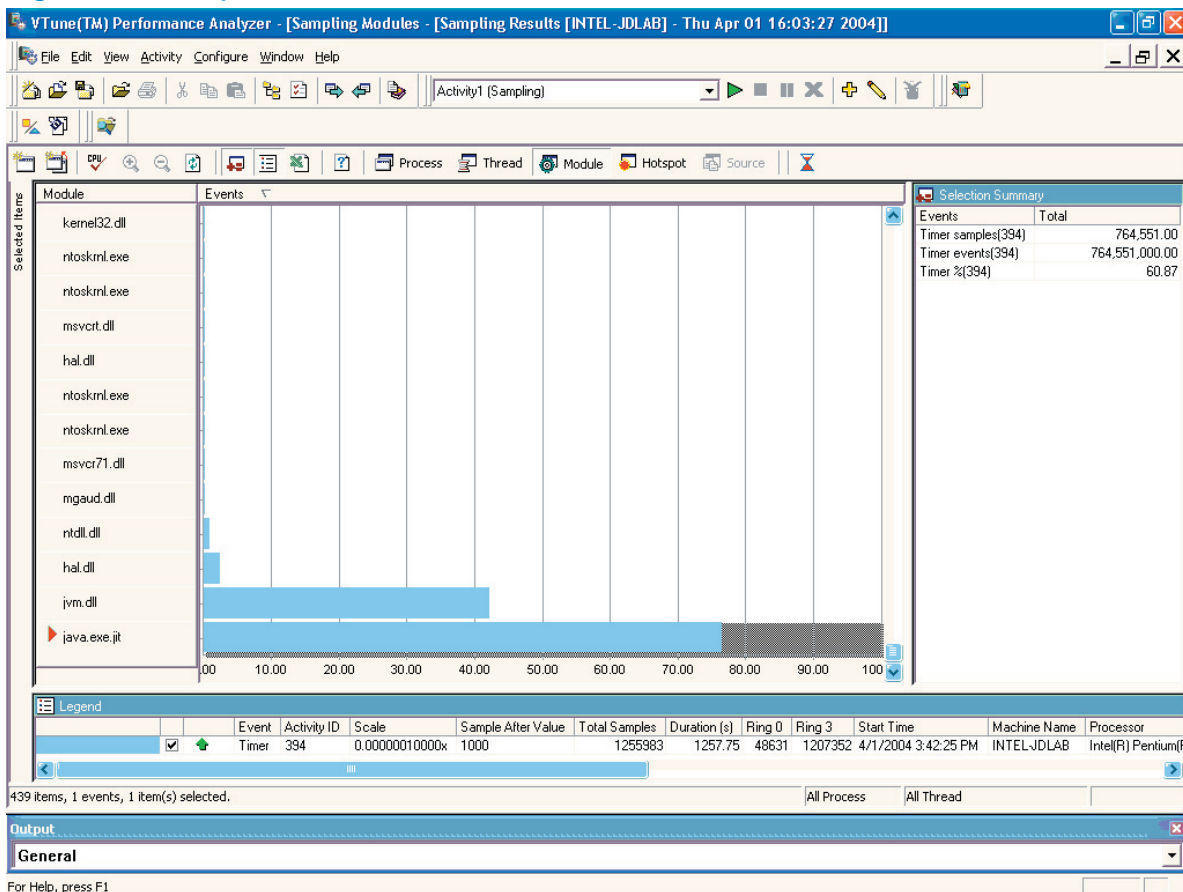
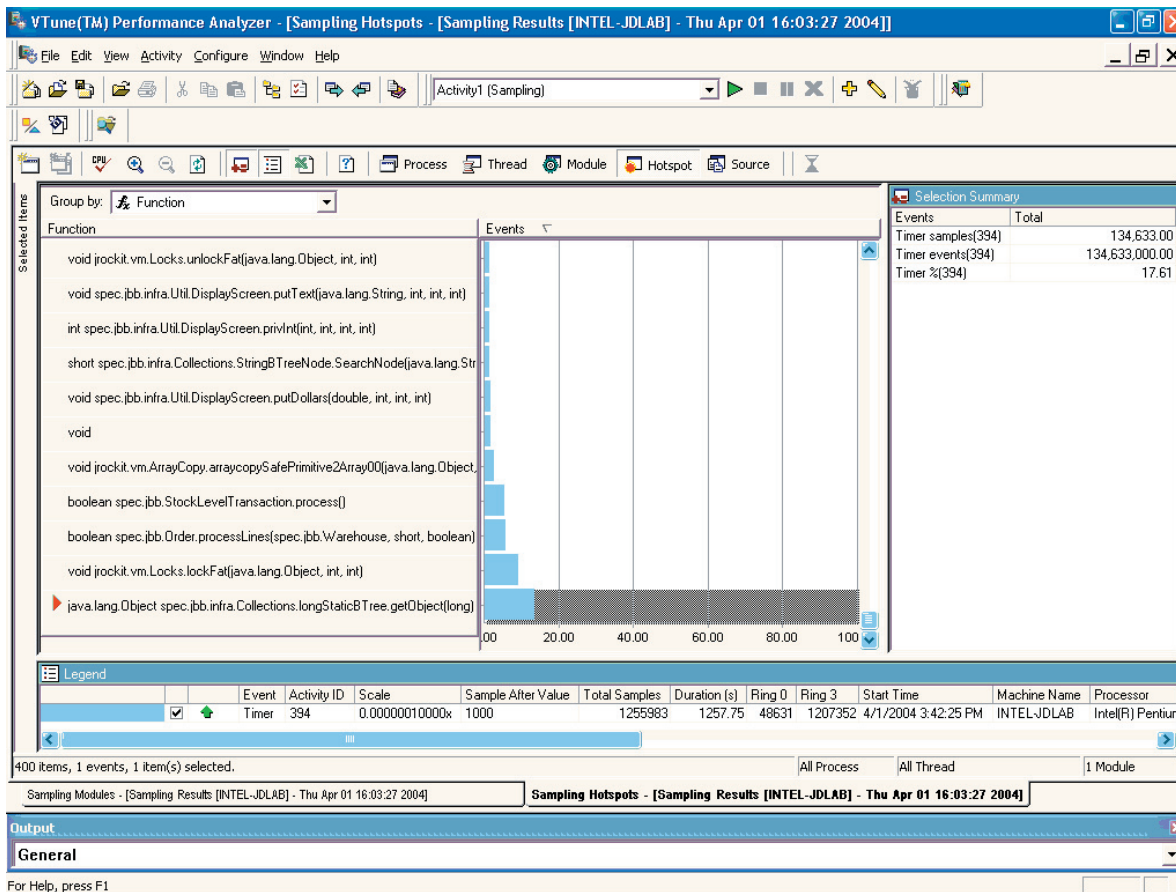


Figure 3 - Function View Example



can modify the source to minimize those events. Minimizing the performance bottleneck events in methods that are frequently called will most likely result in improved application performance.

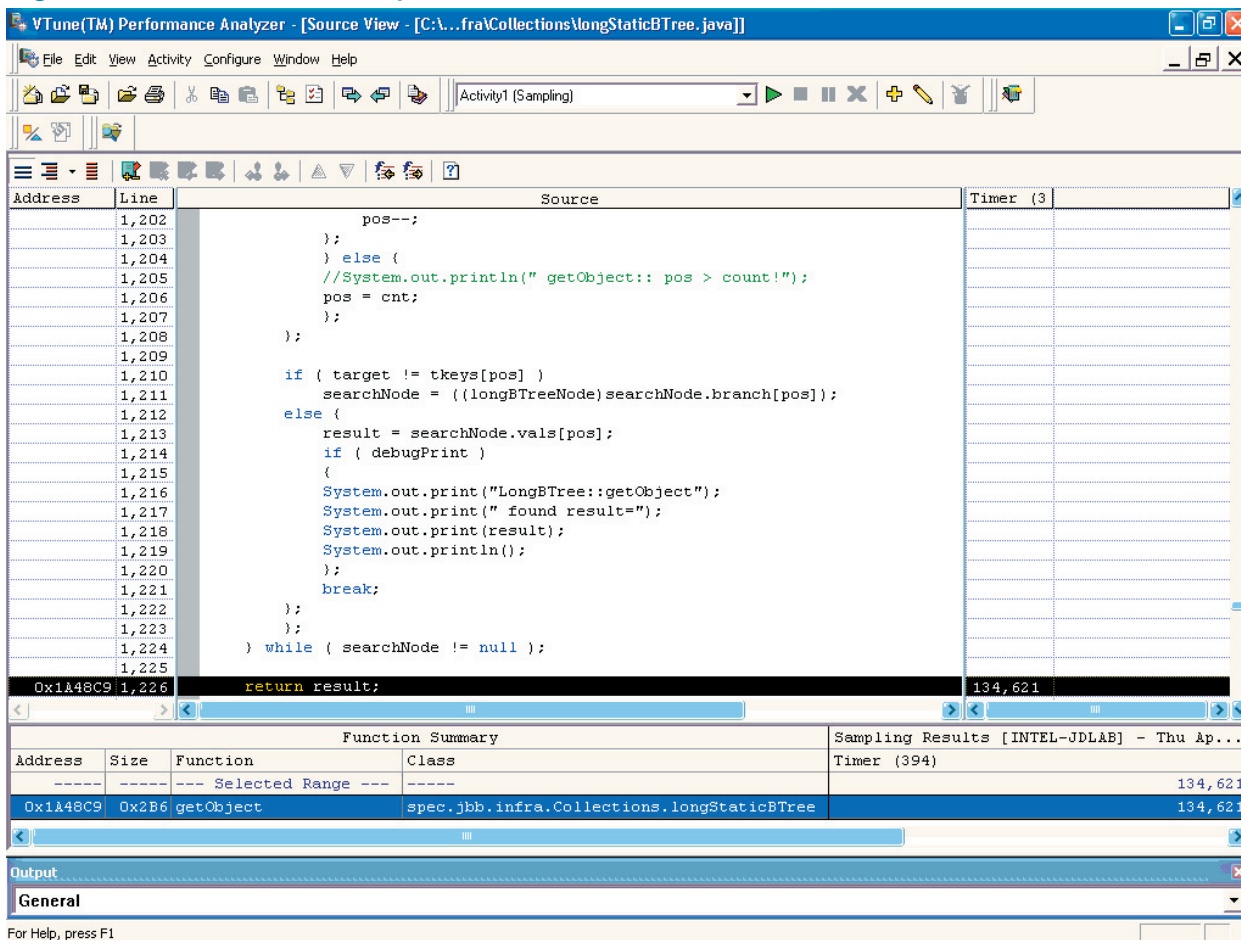
Figure 2 is an example of the “Modules” view for the VTune analyzer Time-Based Sampling technology. Unlike C/C++ code, Java modules are grouped together in a single JIT module. Other modules outside of the JIT module contain native (unmanaged) code. In this example, the analyzer collected samples on the Timer event in the JIT module java.exe.jit. When a user specifies Timer events and a specific sampling interval (e.g., 5 milliseconds), the VTune analyzer sets up all system CPUs to generate an interrupt at every sampling interval. The analyzer records the instruction pointer, process ID and thread ID at each interrupt. The collected information is displayed in graphical format in this view. The graph shows that the java.exe.jit module generated a significant percentage of Timer events.

Double-clicking on the module drills down to the “Function” view for the module, enabling the developer to see additional detail about the code that generated the events.

Figure 3 shows the Function view after double-clicking on the module name from Figure 2. The spec.jbb.infra.Collections.longStaticBTree.getObject method was executing when the largest number of Timer event samples were taken. This method could be a good candidate for tuning. If the Call Graph data indicates that this method is called a significant number of times, then this method is an excellent candidate for code modifications. If source code is available, double-clicking on the method name drills down to the Source View, shown next.

The Source View in Figure 4 shows the area where the greatest number of Timer event samples occurred. This area is also near Loop constructs, and therefore could be a good area to look for optimization opportunities. Once the source has been modified and optimized for the Timer event, the developer could then run additional Sampling activities based upon other processor events such as cache misses, or misaligned data accesses, to tune the code for the specific platform architecture. If the identified bottlenecks occur in the JVM modules, the developer can investigate using another JVM tuned for the platform, or work with the JVM provider to enhance the JVM performance on the platform.

Figure 4 - Source View Example



Summary

The VTune analyzer uses two main technologies and provides a wealth of information that enables the Java developer to locate and tune specific parts of the application that cause performance bottlenecks. The two technologies, Time- and Event-Based Sampling and Call Graph, are directly applicable to Java applications, in addition to C/C++ applications. They enable the VTune analyzer to annotate Java source code with performance data and to display algorithmic, flow-of-control information for Java applications. Call Graph identifies methods and algorithms that are candidates for tuning, and Time- and Event-Based Sampling and Drill Down identify specific source lines. Once these candidates are identified, only the specific parts of the application which consume significant amounts of time need to be tuned, enabling Java application developers to use the Java APIs efficiently and to use the best JVM for the platform.

More information on the VTune Performance Analyzer is available in the Getting Started Tutorial (available in the VTune Analyzer under Help > GettingStartedTutorial, or on the Web at <http://www.intel.com/software/products/college/vtune/getstart/tutorial/index.htm>) or by simply using the VTune analyzer's extensive help system.

Supported Java* Environments and OS Platforms

The VTune analyzer supports BEA JRockit* 8.x, IBM JDK* 1.3 or newer, and Sun JDK* 1.4.x. VTune analyzer with Java profiling capabilities is currently available on various Windows operating systems running on IA-32 and the Intel Itanium® processor family architecture hosts, as well as Linux* for IA-32.

Additional Resources

Here are some locations for additional information:

Overview of VTune analyzer features:

<http://www.intel.com/software/products/vtune/vpa/overview.htm>

Supported environments:

<http://www.intel.com/software/products/vtune/vpa/sysreq.htm>

Free VTune analyzer evaluation application for download:

<http://www.intel.com/software/products/global/eval.htm>

VTune analyzer Flash demo:

http://www.intel.com/software/products/vtune/downloads/VTune_V6.htm

General information on the VTune analyzer:

<http://www.intel.com/software/products/vtune/>

General information on Intel® Software Development Products:

<http://www.intel.com/software/products/>

Intel Software Training:

<http://www.intel.com/software/college>



Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052-8119
USA

For product and purchase information visit:
www.intel.com/software/products

Intel, the Intel Logo, Itanium, Pentium, Intel Xeon, Intel XScale and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.